2004

On Time Synchronization in Multi-hop Sensor Networks

Ossama Younis

Sonia Fahmy
Purdue University, fahmy@cs.purdue.edu

Report Number:
04-020

ON TIME SYNCHRONIZATION IN MULTI-HOP SENSOR NETWORKS

Ossama Younis
Sonia Fahmy

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #04-020
June 2004

On Time Synchronization in Multi-hop Sensor Networks

Ossama Younis and Sonia Fahmy
Department of Computer Sciences, Purdue University
250 N. University Street, West Lafayette, IN 47907–2066, USA
e-mail: {oyounis,fahmy}@cs.purdue.edu

Abstract— **Time synchronization is essential for several ad-hoc network protocols and applications, such as TDMA scheduling, data aggregation, caching, object tracking, and security checking. Prior work on synchronization in wireless networks has not adequately addressed rapid convergence and scalability requirements in dense networks serving time-sensitive applications, such as sensor networks. In this paper, we propose a distributed clustering-based high-level time synchronization framework for multi-hop ad-hoc networks that builds a two-tired, synchronized network. We do not make any assumptions about node capabilities (e.g., being GPS-enabled), or the presence of reference nodes in the network. Thus, global consensus on one time value is not our goal. Rather, we assume that relative node synchronization is sufficient. We study both classes of sensor network applications, namely, source-driven and data-driven applications. We give fully distributed protocols for regional synchronization (nodes within 2-hops), path synchronization, and global (inter-cluster) network synchronization. Our proposed path synchronization protocol (SYNC-PATH) is reactive, while our inter-cluster network synchronization protocol (SYNC-NET) is proactive. The protocols exploit the fact that for most applications, coarse-grained accuracy is sufficient at the global scale. Our framework is independent of the clustering and inter-cluster routing approach, and the underlying low-level synchronization protocol, and thus is suitable for use in conjunction with both receiver-receiver and sender-receiver synchronization approaches. We analyze each protocol and prove that it terminates in $O(1)$ time. We also provide a density model for validating SYNC-NET, and evaluate all protocols via extensive simulations. Our framework can be employed in any ad-hoc wireless network setting.**

Index Terms— **System design, simulations, multi-hop ad-hoc networks, sensor networks, time synchronization, node clustering**

– This research has been sponsored in part by NSF grant ANI-0238294 (CAREER) and the Schlumberger Foundation technical merit award.

I. Introduction

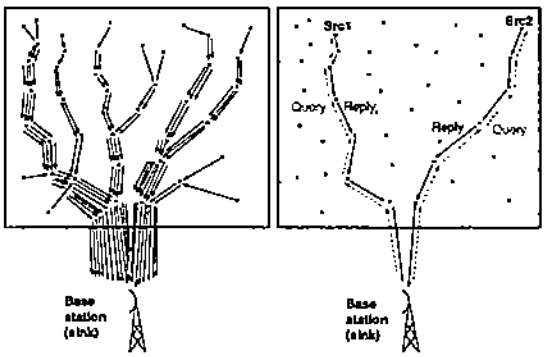
Rapid advancements in sensor network technology have provided incentives for research on sensor protocols and services. Sensor nodes are typically left unattended, rendering it infeasible to re-charge their batteries or synchronize their clocks. Time synchronization, however, is critical for several applications, including sensor network applications. For example, data aggregation operations require timing information to combine events which occur within specified time frames. Applications that exploit caching also need timestamping to avoid adding stale (or duplicate) information to the cache tables. TDMA scheduling requires accurate knowledge of time lags and continuous synchronization among participating nodes to avoid interference. For secure communication, nodes typically use symmetric-key cryptography for maintaining secure channels. This requires periodic key re-distribution that is based on time triggers to avoid possible cryptanalysis. For example, μTESLA [1] is an energy-efficient protocol proposed for message authentication in sensor networks. μTESLA requires the network nodes be fully synchronized. Several cryptographic schemes for ad-hoc networks also require that timestamps be included as part of the digital signature for validation. Lack of knowledge about the relative synchronization among nodes in this case can lead to erroneous conclusions. Although digital signatures are not common for sensors, they can be used in any non energy-constrained ad-hoc network setting, for which Our framework is also suitable. Other time-sensitive applications that require synchronization include object tracking and navigation guidance.

Research on time synchronization in networked distributed systems has proceeded in two directions. In the first direction, synchronization is based on virtual clock ordering. This is sufficient in systems where absolute timing is not necessary (i.e., event ordering is sufficient) [2]. In the second direction, physical clock synchronization is performed for applications sensitive to

absolute timing, e.g., for data aggregation. The Network Time Protocol (NTP) [3] has been very successful in synchronizing the Internet and belongs to the latter direction. For wireless ad-hoc networks, the Global Positioning System (GPS) has provided a solution for node synchronization. A node equipped with a GPS antenna can synchronize its clock with a satellite. Coarse granularity is easily achieved by setting the receiver clock to that of the GPS-equipped initiator timestamp when a synchronization pulse is received. Several protocols use this approach and assume that a few GPS enabled nodes are available in the network to act as initiators.

Time synchronization in sensor networks faces three challenges, namely, energy-scarcity, hardware cost, and dense deployment. The foremost challenge is the energy-efficiency requirement, which makes the use of energy-consuming devices, such as GPS, inappropriate. Energy-efficiency also entails using low overhead protocols, which may trade off accuracy for network longevity and fast convergence. The cost of adding hardware devices for clock synchronization (such as GPS) is usually very high compared to the price of the sensor itself, and thus adding redundant nodes to synchronize the network might be more cost-effective [4]. Dense deployment of sensor nodes necessitates the design of scalable solutions. The application requirements and environmental conditions impose challenges on any proposed synchronization framework. For example, if the application requires fine-granularity, or if the clocks drift within short time periods, then node synchronization has to be invoked frequently.

Sensor network applications can be classified into two main categories: source-driven and data-driven. In source-driven applications, nodes periodically send reports to an observer (e.g., a base station) about a measured parameter(s). Figure 1(a) depicts a source-driven network where all the nodes report their readings to a base station via a multi-hop routing path. Environmental monitoring is an example of source-driven applications, e.g., monitoring the temperature, radiation, or chemical activity within a certain field or plant. In data-driven applications, an observer queries the network about information of interest, and a source node in possession of this information replies with an answer. An example data-driven application is an object tracking system, in which an observer queries the network about the occurrence or behavior of an object. Figure 1(b) depicts a data-driven network where the sink sends two queries that are answered by two different nodes. Note that queries are typically flooded unless the sink knows the exact sensor which has the answer it is looking for. We only show the query/reply paths and ignore the rest of the flooded traffic.



(a) Source-driven network (b) Data-driven network

Fig. 1. Examples of sensor network classes

Several protocols have been proposed for time synchronization at the application level [5], [6], [7], [8], [9], [10]. Performing synchronization at layers higher than the network layer gives more flexibility to application needs and allows for more energy-efficiency. Time synchronization can be classified as low-level and high-level synchronization. Low-level synchronization involves the physical process of synchronizing two or more clocks within a region. Example protocols are [5], [8], [11], [9], [6]. Low-level time synchronization can further be classified into sender-receiver and receiver-receiver approaches. In sender-receiver approaches, such as TPSN [8], a receiver adjusts its clock according to the timestamp received from a sender. In receiver-receiver approaches, such as RBS [5], receivers use sender synchronization pulses to synchronize among themselves (by timestamping these pulses and exchanging these timestamps). The sender is not synchronized with them in this case. Receiver-receiver (RR) synchronization has three advantages over sender-receiver (SR) synchronization for sensor network applications. First, RR synchronization does not require the presence of GPS-enabled nodes in the network to act as reference nodes. Second, RR approaches give higher accuracy than SR approaches if timestamping is not possible at the MAC layer. Third, even if MAC layer timestamping is possible, it is not reasonable for a node to follow the clock of another node which does not have a reference time clock. This occurs in systems where all nodes have similar capabilities and none has any special equipment, such as GPS. SR synchronization, however, has negligible message exchange overhead as compared to RR synchronization. In contrast to these low-level approaches, high-level synchronization describes how an entire path or network is synchronized, regardless of the underlying protocol used to physically

synchronize the clocks. Example protocols are given in [2], [10], [7], [6] (multi-hop TPSN [8] and multi-hop RBS [5] also belong to this category).

The above approaches for high-level synchronization, however, have not adequately considered several factors that may hinder their application in multi-hop wireless networks. These factors include: (1) rapidly synchronizing the network (i.e., in $O(1)$ time) since fast response is important for several applications, especially when the network is dense and the synchronization algorithm has to be invoked frequently (whenever the network goes out-of-sync), (2) flexibility in selecting the appropriate approaches to use according to the type of application and expected frequency of queries, (3) defining synchronization regions and electing synchronization initiator(s) in the network, since not all the nodes lie within the same broadcast range of each other (especially for those on the boundary), (4) minimizing the message overhead of the synchronization process, and (5) scalably performing multi-hop synchronization, even if the synchronization regions are not intersecting. Even for intersecting regions, the routing protocol has to select synchronized paths for forwarding data. Thus, we need to view the network as a "time-aware" graph to ensure "time" connectivity. At the same time, the synchronization process must be transparent to the underlying routing protocol.

In this work, we consider the above challenges that must be addressed to achieve fast high-level synchronization at the global scale. We propose a new framework for time synchronization in multi-hop sensor networks that integrates synchronization with node clustering to construct two-tiered, synchronized networks. Our framework provides efficient and flexible synchronization mechanisms to serve different types of applications (source-driven and data-driven) for different network loads (number of queries and reports). Our framework is independent of the clustering and inter-cluster routing approach, and the underlying low-level synchronization protocol. We will consider the more demanding scenario for low-level synchronization, namely, the receiver-receiver approach, which provides fine-grained synchronization and does not assume any infra-structure support. To the best of our knowledge, our proposed framework for high-level time synchronization is unique in its underlying assumptions, objectives, and methodology, compared to previous work. More specifically, the diffusion-based method [10] and TPSN [8], proposed for global synchronization, assume an application that requires global network agreement on one clock value. This requirement is too strict for most applications, since local knowledge of clock differences is typically sufficient for translation of reported time values. In addition, we do not assume the presence of any special node(s) in the network as in [8]. Our goal is end-to-end synchronization of any querying/receiving entity and any node in the network, and not common time consensus among all network nodes. We propose high-level protocols suitable for use with both receiver-receiver and sender-receiver low-level synchronization mechanisms.

The remainder of this paper is organized as follows. Section II defines the terms used throughout the paper, and outlines the problem addressed in this work. Section III provides the design rationale and approach used. It also presents algorithms for intra-cluster and inter-cluster synchronization, and argues that they satisfy our goals. Section IV evaluates the proposed algorithms via simulation. Section V discusses design and deployment issues in our framework. Section VI briefly surveys related work. Finally, Section VII summarizes our work and suggests directions for future research.

II. Problem Definition

In this section, we define new terms and functions that will be used throughout this paper. Later, we will formulate the problem that we address by outlining the system model and the goals of our proposed framework.

A. Definitions

We define the function *SYNC*, the types of node synchronization, and the notion of a synchronized path.

Definition 1: For any two nodes u and v, the function SYNC(u,v) = 1 if v is synchronized with u; and SYNC(u,v) = 0 otherwise. SYNC(u,v) is a transitive function, i.e., if SYNC(u,v) = 1 and SYNC(v,w) = 1, then SYNC(u,w) = 1.

Definition 2: Nodes u and v are said to be *physically synchronized* if $|clock(u) - clock(v)| \leq \epsilon$, where ϵ is the target accuracy. This type of synchronization is symmetric since SYNC(u,v) = SYNC(v,u).

Definition 3: Nodes u and v are said to be *relatively synchronized* if one of them (or both) is aware of the difference $|clock(u) - clock(v)|$. This type of synchronization is asymmetric since SYNC(u,v) = $i \not\rightarrow$ SYNC(v,u) = i, where $i = 0, 1$.

Definition 4: A *strictly synchronized path* $P(v_1, v_{|P|})$ is an ordered set of nodes between a source v_1 and a destination $v_{|P|}$, such that either SYNC($v_1, v_{|P|}$) = 1, if $|P| = 2$, or $\forall v_i \in P$, SYNC(v_{i-1}, v_i) = SYNC(v_i, v_{i+1}) = 1, where $1 < i < |P|$.

Definition 5: A *loosely synchronized path* $P(v_1, v_{|P|})$ is an ordered set of nodes between a source v_1 and a destination $v_{|P|}$, such that either SYNC($v_1, v_{|P|}$) = 1, if $|P| = 2$, or $\exists i, j : 1 < i, j < |P|$, such that $v_i, v_j \in$

$P(v_1, v_{|P|})$, $j \geq i$, SYNC$(v_1, v_i) = 1$, SYNC$(v_j, v_{|P|}) = 1$, and the path $P(v_i, v_j)$ is loosely synchronized.

In other words, a strictly synchronized path is one in which every two adjacent nodes on the path are synchronized. The definition of a loosely synchronized path is recursive. It requires that a node on the path is synchronized with the source and another node on the path is synchronized with the receiver, and the path among these two nodes is also loosely synchronized. This is usually sufficient for query-driven applications. For example, assume a military application, where a soldier (observer) floods a query asking if any node has sensed a moving tank, and at what time. One or more nodes (senders) around the network will reply positively and report their timestamps. The soldier device should be able to interpret these timestamps according to its clock. Synchronization with *all* the nodes on the paths from the senders to the soldier is not useful in this case. However, if data aggregation occurs on different paths to the soldier, then strict path synchronization is required. Figure 2 gives an example of strictly versus loosely synchronized paths.

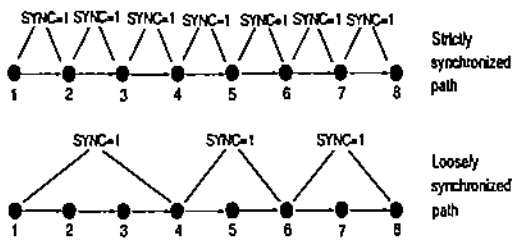


Fig. 2. An example of strictly versus loosely synchronized paths

Definition 6: Let V denote the set of nodes in the network, and B denote the set of observers. A *synchronized network* is one in which $\forall v \in V$, and $\forall b \in B$, $\exists$ at least one synchronized path $P(v, b)$.

B. System Model

We use a general model for ad-hoc sensor networks and do not assume any infrastructure support. This is important for the generality and robustness of the proposed framework, since in environments with malicious users, attacks can be launched at more highly equipped nodes, e.g., nodes with GPS antennae.

Assume that n sensors are dispersed in a square field of side length L. Assume the network has the following properties:

- The network is quasi-stationary.
- Each node has a unique identifier and a fixed number of transmission power levels.
- Nodes are left unattended after deployment and are location unaware.

We also make two assumptions related to the synchronization process:

- Any two neighboring nodes can be synchronized in $O(1)$ time.
- A synchronization initiator node (one that generates synchronization pulses) can synchronize its neighbors, but cannot be synchronized with them unless its neighbors will directly copy its reported clock value (i.e., the underlying low-level synchronization mechanism is allowed to be receiver-receiver).

The first assumption is reasonable since two nodes can typically be synchronized by exchanging a fixed number of messages and averaging the delay. There is no guarantee in this case, however, about the fine granularity of synchronization if messages can be lost or delay can be unbounded [12]. The second assumption makes it possible to utilize receiver-receiver, fine-grained, low-level synchronization mechanisms, such as RBS [5]. If the network does not rely on infrastructure support, then a synchronization initiator is just an arbitrary node. Therefore, its neighbors are not likely to copy its clock value, but will just use it to synchronize among themselves. This is in contrast to mechanisms such as TPSN [8] which assume the existence of at least one reference node (e.g., equipped with a GPS antenna), and thus builds a hierarchy to synchronize the network using a sender-receiver approach.

C. Goals

The ultimate goal of this work is to provide a framework for fast and scalable time synchronization in ad-hoc networks. The framework suggests approaches and mechanisms to use for data-driven and source-driven networks, according to the expected offered load. The synchronization process should be of $O(1)$ time complexity. We focus on relative synchronization in this work which is sufficient for sensor network applications that do not rely on any infrastructure. Our framework will provide mechanisms for synchronizing a region, a path, or an entire inter-cluster network. A region R in the network can be defined as follows. Any two nodes $u, v \in R$ can reach each other in either: (1) one hop, or (2) two hops through a node w, such that $w \in R$.

In other words, we design mechanisms to support the following requirements:

- *Regional synchronization:* Assume that $\exists$ a node $w \in R$, such that $\forall v \in R$, $distance(v, w) = 1$. Then, $SYNC(v, w) = 1$ (R is a region in the network).
- *Relative path synchronization:* For any query Q issued by an observer v_1 and answered by a source

$v_{|P|}$, $\exists$ at least one loosely synchronized path $P(v_1, v_{|P|})$.

- *Relative global synchronization:* For a multi-hop network with a set V of nodes, $\exists$ at least one strictly synchronized path P from any $v_i \in V$ to the observer(s).

Note that our goal is not to achieve a certain synchronization accuracy between the sender and the receiver, or bring the network to a common time consensus. We are merely interested in rapid relative synchronization between senders and receivers to the best that the underlying low-level synchronization mechanism can provide. We find loose synchronization useful for applications that only require the ability of the sink to translate a timestamp obtained from a source. Strict synchronization, on the other hand, is useful for any scenario where synchronization is required at every node on a path from sources to sinks, e.g., in data aggregation. The overhead of translating the timestamp in a packet as it is forwarded on a path is cheap since a node only holds the relative time differences with a limited set of neighboring nodes (as described later).

III. A Time Synchronization Framework

A. Design Rationale and Approach

The expected network load is determined by the network application. The application must therefore select the appropriate synchronization mechanism according to its offered load and the locality of generated queries. For example, for low to moderate loads, the network does not need be globally synchronized. Rather, synchronization on the forwarding path is typically sufficient. This type of synchronization can be performed "reactively" or "on-demand" (we borrow these terms from the routing literature). In contrast, global synchronization is a process that involves all network nodes and must be periodically performed in a heavily-loaded network where queries do not follow a distinct locality pattern or the application is sender-driven. We refer to this type of synchronization as "proactive synchronization." Data-driven networks can typically exploit locality of requests more than source-driven networks, unless the observer is mobile and its location changes significantly between the issuance of queries. We do not make any assumptions about the mobility of observers in our work. We provide algorithms that are suitable for stationary and mobile observers, and leave the choice of the appropriate algorithm to the application. Table I summarizes these options for different network applications.

A number of protocols, such as multi-hop RBS [5], assume the network to be divided into intersecting regions and rely on nodes in the intersection areas to propagate synchronization information as data is forwarded. A region in this context is an area in which single-hop communication is possible between every pair of nodes. To understand the problems caused by non-intersecting regions in the network, consider the scenario in Figure 3. In this scenario, the network is divided into three regions around nodes A, B, and C. These regions are non-intersecting (because no nodes lie in their intersection regions). Therefore, a packet sent from node 1 to node 8 will not find a synchronized path although this would have been possible if nodes 2, 5, and 7 were the synchronization initiators. This problem depends on node density, node distribution, and transmission range. Therefore, the network has to be organized such that regions are clearly defined and inter-regional communication is possible, even if regions are non-intersecting.

TABLE I

Selecting the appropriate synchronization approach according to the type of application and offered load

	Data-driven network	**Source-driven network**
Low load	On-demand path synchronization	– On-demand path synchronization for localized requests – Proactive global synchronization for distributed requests
High load	– On-demand path synchronization for localized requests – Proactive path synchronization for distributed requests	Proactive global synchronization

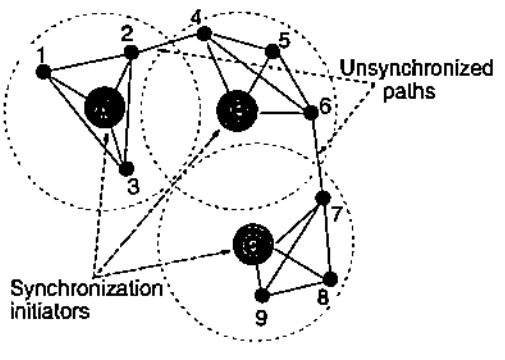


Fig. 3. Multi-hop network with three regions. The figure demonstrates the failure to find inter-regional synchronized paths

Node clustering increases scalability. In a clustered network, a number of nodes act as cluster heads, and communicate with their cluster nodes (intra-cluster communication), and their neighboring cluster heads (inter-

cluster communication) or with the observer. To achieve inter-cluster connectivity, a clustering protocol can use a small to medium transmission range (power level) for building clusters, reserving higher ranges for inter-cluster communication or communication with the observer. We assume that a node subscribes to only one cluster. Node clustering has been used for routing [13], [14], for improving the network capacity [15], for supporting data aggregation, and for prolonging the network lifetime by distributing load among network nodes [16], [17]. For node synchronization, clustering can play an important role in: (1) defining synchronization regions (clusters), by selecting the appropriate cluster power level, (2) selecting the synchronization initiators in the network (e.g., cluster heads), (3) adapting to application requirements by expanding or contracting the synchronization regions (cluster sizes), (4) increasing the synchronization accuracy by synchronizing 2-hop neighbors through cluster heads, and (5) enabling scalable and efficient multi-hop synchronization by synchronizing each cluster independently and only relying on the cluster head overlay (i.e., the inter-cluster head network) for synchronizing the network and propagating time information. This reduces the message overhead, increases the perceived accuracy, and increases scalability.

Typically, the application selects a power level R_c for cluster formation and intra-cluster communication according to the node capabilities and MAC protocol, node density, and transmission patterns, in order to maximize spatial reuse and reduce energy consumption. The selection of the best cluster power level is beyond the scope of this work. Our main concern is that the cluster head overlay is connected. This can be achieved if the relation between the number of nodes in this overlay, n_0, and the maximum transmission range $R_m > R_c$ (which can be used for inter-cluster communication) of a node satisfies the connectivity conditions specified in [18]. That is, assuming that a node is active with probability p, the necessary condition for connectivity and coverage is $R_m^2 \geq \frac{c \log n_0}{p n_0}$, where $c = \frac{1}{\pi\beta^2}$, and $\beta \leq 0.5$ (this is a generalization of the result in [19]). We will define our density model in Section III-D to provide the necessary conditions for connectivity and coverage in our global synchronization algorithm.

Our synchronization approach is independent of the clustering protocol and the inter-cluster routing protocol used. Ideally, the clustering protocol should ensure that cluster heads are well-distributed in the network area. For example, cluster heads can be considered well-distributed if no two cluster heads can communicate using the cluster range R_c, which is only sufficient for intra-cluster communication. This allows multiple synchronization operations to proceed simultaneously in the network without interference. Note that in [10], node clustering was proposed to scale down the complexity of the proposed centralized synchronization scheme (however, the asymptotic time and message complexities remain the same). In our work, node clustering plays an important role in organizing the network for facilitating the synchronization process.

We now present a number of algorithms for intra-cluster and inter-cluster synchronization. For intra-cluster synchronization, cluster members are synchronized with their cluster heads. For inter-cluster synchronization, nodes in the cluster head overlay are synchronized, independent of the rest of the network. The inter-cluster synchronization algorithms can be limited to the path of the querying observer (path synchronization), or can cover the entire clustered network (global synchronization). Loose path synchronization is useful to applications requiring correct translation of a timestamp sent by a source at its destination. An example application is environmental monitoring, where a source sends a timestamped temperature when it exceeds a threshold. Strict inter-cluster synchronization is required if data aggregation or caching is performed at every node along the path. The transmitted packet does not increase in length since at every hop on the path the timestamp value is translated and only the new one is carried on to the next hop.

B. Intra-cluster Synchronization (SYNC-IN)

For intra-cluster synchronization, all nodes within a cluster only need to be synchronized with the cluster head. The cluster head cannot synchronize itself with the cluster nodes if it acts as a synchronization initiator (assuming receiver-receiver low-level synchronization) as stated in Section II-B. The cluster head thus elects nodes from within its cluster to act as initiators, one at a time. It continues doing so until all the nodes subscribed to its cluster are synchronized with the cluster head. Since this is an intra-cluster operation, and to avoid interference with neighboring clusters, pulses (messages) for intra-cluster synchronization are sent using the cluster range R_c (i.e., the power level used for cluster formation). This also increases energy efficiency. Figure 4 shows pseudo-code for the intra-cluster synchronization algorithm executed at each cluster head.

The best candidate to select as a synchronization source is one which is closest to the cluster head. This is because a close neighbor is likely able to cover most of the nodes in the cluster using the cluster range. A cluster head can maintain lists of neighbors using each of its

Fig. 4. SYNC-IN: Intra-cluster Synchronization Algorithm

// Let S = ϕ, Cluster = C, Cluster head = CH
// Range is given as an input parameter
1. $V_c \leftarrow \{v : v \in C,\ v \neq CH\}$
2. WHILE $|S| < |V_c|$
3. Pick $u \in (V_c - S)$ as synchronization initiator
4. Send S to u
5. IF ($\nexists v \in S$, such that $v \in$ neighbor(u) and $v \neq CH$)
6. Synchronize(u, CH)
7. ELSE
8. Apply low-level synchronization using u
9. $S \leftarrow$ S $\cup$ {v: SYNC(v,CH) = 1}

available power levels, so that neighbors in the smallest level are closest (this idea was used in CLUSTER-POW [15] for minimum power communications). If the cluster head cannot deduce the proximity of its cluster members, random selection can be applied. During the operation of this protocol, nodes are synchronized with the cluster head and removed from the candidate set of synchronization initiators. If the selected initiator does not have any neighbors that are yet to be synchronized with the cluster head, it directly synchronizes itself with the cluster head. This can be done in $O(1)$ time using techniques such as [12], We now prove the correctness of this algorithm. We also show that the maximum number of nodes required to act as initiators and fully synchronize the entire cluster with its head is constant, and thus fast intra-cluster synchronization is achieved.

Lemma 1: When the SYNC-IN algorithm terminates, all nodes in the cluster are synchronized with the cluster head, CH.

Proof. We first show that $|S_{i+1}| > |S_i|$, where S_i and S_{i+1} are the sets of nodes synchronized with CH at the beginning of iterations i and $i+1$, respectively. At iteration i, CH picks a node $u \notin S_i$ to act as a synchronization initiator. This results in at least one newly synchronized node(s) that was not in S_i. Thus, $|S_{i+1}|$ is strictly larger than $|S_i|$. The algorithm only terminates when $|S| = |V_c|$. □

Lemma 2: The SYNC-IN algorithm terminates in $O(1)$ iterations.

Proof. The algorithm terminates when all the nodes in the cluster are visited as either initiators or receivers. The number of iterations depends on the portion of the cluster that is covered each time a node is elected to act as an initiator. The worst case scenario is demonstrated in Figure 5 where the elected nodes are very close to the boundary of the cluster, i.e., on the perimeter of the virtual transmission circle of the cluster head.

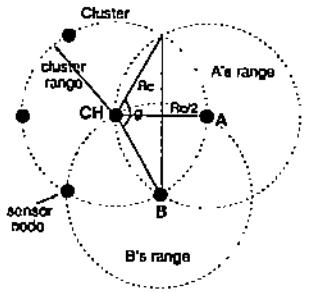


Fig. 5. Worst case scenario for electing initiators to perform intra-cluster synchronization

Assuming that the cluster circle has a perimeter p, the length of the arc covered in circle CH by circle A is $p/3$. This is because the opposite angle g is $2\pi/3$ (since $cos(g/2) = 0.5$). In the worst case, the next elected node B is also on the perimeter of CH and A. This covers another arc of CH of length $\pi/3$. We can add at most three other nodes on the perimeter of CH to cover the entire area of CH. Therefore, the SYNC-IN algorithm requires at most 5 iterations to visit all "non-initiator" nodes and at most another 5 iterations to visit each of the initiators again. SYNC-IN terminates in at most 10 iterations (we will verify this result in Section IV). An "iteration" in this context denotes a low-level synchronization process of a group of nodes in the cluster. Interference is also reduced since only one node transmits synchronization pulses at any time. □

Lemma 3: The SYNC-IN algorithm requires $O(1)$ message transmissions per node in the cluster.

Proof. A node participates in the synchronization process in only one iteration and sends only one message. A node may act as an initiator only once. Thus, each node (other than the cluster head) sends at most two messages during the entire synchronization process (i.e., $O(1)$). The cluster head sends two messages at each iteration (one to elect an initiator and one for synchronization). Since the number of iterations is $O(1)$, the cluster head also sends $O(1)$ messages. □

The synchronization initiator may actually send $O(1)$ synchronization pulses, and not just one. This, however, does not affect the validity of the above proof. It is also worth mentioning that for sender-receiver synchronization, intra-cluster synchronization will be much simpler. This is because the cluster head can act as an initiator in this case, and the entire process terminates in only one iteration. The message overhead is still within the same bounds as in receiver-receiver synchronization. Finally, note that SYNC-IN takes the transmission range as an input parameter, and therefore it can synchronize the cluster head with nodes outside the cluster range if the specified transmission range is larger.

C. Inter-cluster Path Synchronization (SYNC-PATH)

As discussed in Section III-A, path synchronization involves only the nodes in the cluster head overlay on the path from a source to a destination. The return path P can be determined using a routing protocol, such as Directed Diffusion [20]. P can also be determined on the fly if data is forwarded from the source to the destination using routing tables. In the former case, P is known before data is sent from the source, and thus synchronization can take place while data is being forwarded. In the latter case, a synchronization trigger packet can be sent prior to data transmission to record nodes on the path and trigger synchronization at "some" of these nodes (as explained below). In either case, each node is aware of all the preceding nodes on the path P, and synchronizes with at least one of them. The action taken by our algorithm is similar in essence to the post-facto protocol [7] since they are both "reactive" approaches. However, the post-facto protocol is intended for regional synchronization of nodes that collaborate to report an event or stimulus and assumes that all nodes are within communication range of each other. We consider longer paths that require multi-hop communication, and assume that a low-level synchronization protocol will be used among every 2-hop neighbors.

The algorithm proceeds as follows. Assume that node $u \in P$ ($u \neq v_{|P|}$) is the one currently executing the SYNC-PATH algorithm. Node u is aware of the preceding nodes in P (call them P'). Assume that $u_1 \in P$ is the node directly preceding u (thus, $u_1 \in P'$). If u is synchronized with any $u' \in P'$ (not necessarily u_1), then it simply forwards the packets (data or synchronization trigger) to the next hop $u_2 \in P$. Otherwise, u decides to act as a synchronization initiator and applies a modified version of SYNC-IN (SYNC-2HOPS) using a transmission range R_t, where R_t is the transmission range used for inter-cluster communication. Only nodes in the cluster head overlay that are within R_t distance from u are involved in the SYNC-2HOPS algorithm execution. The SYNC-2HOPS algorithm uses u as a synchronization initiator. The node u is also responsible for collecting the timestamp information from its neighbor cluster heads (within inter-cluster transmission range R_t), computing the relative synchronization for all of its neighbors, and sending this information back to them. The latter case results in u_1 and u_2 getting synchronized since they are both cluster head neighbors of u. This process continues until the destination is reached.

A side effect of applying SYNC-2HOPS with transmission range R_t in the cluster head overlay is the synchronization of all neighbor cluster heads of u with each other. This may be useful if the application requests follow a distinct pattern. This approach can also be implemented in the low-level synchronization protocol used. For example, if RBS is used, then u can act as an initiator, collect the readings from u_1 and u_2 (which may not hear each other), and send them back their relative synchronization information. Figure 6 gives pseudo-code for Algorithm SYNC-PATH.

Fig. 6. SYNC-PATH: Inter-cluster Path Synchronization

```
// This algorithm is executed at node u ∈ P
// Let P' be the set of nodes on P preceding u
1.  P' = {v : v ∈ P, v is a predecessor(u)}
2.  IF ∄v ∈ P', such that SYNC(u, v) = 1
3.     SYNC-2HOPS(R_t)
4.  Forward the synchronization trigger packet
```

Lemma 4: The path P generated by SYNC-PATH between a source v_1 and a destination $v_{|P|}$ is loosely synchronized.

Proof. We prove this lemma by contradiction. Assume that the path P is not loosely synchronized. Without loss of generality, assume that $\exists v_k \in P$, $1 < k < |P|$, such that the path $P'(v_1, v_k)$ is loosely synchronized, while the path $P''(v_k, v_{|P|})$ is unsynchronized. v_k is synchronized with one of its predecessors on P'. When v_k executes SYNC-PATH, it just forwards the synchronization trigger packet to v_{k+1}. Now, v_{k+1} finds itself unsynchronized with v_k, and consequently all P'. SYNC-PATH forces v_{k+1} to act as a synchronization initiator and execute SYNC-2HOPS(R_t). Consequently, $SYNC(v_k, v_{k+2})$ becomes 1. Therefore, the path $(v_1, v_k + 2)$ is loosely synchronized. This continues throughout P'' until $v_{|P|}$ is reached. Thus P is loosely synchronized. □

For odd length paths, the last cluster head on the path will have to directly synchronize with the observer. It is obvious that SYNC-PATH reduces the propagated error compared to a straightforward approach in which every pair of adjacent nodes on the path are synchronized. This is because only every other node is synchronized using SYNC-2HOPS, and not every pair of adjacent nodes. For example, synchronization of a path with h hops and σ_h per-hop error is expected to result in an $\sqrt{h}\sigma_h$ total path error using multi-hop RBS [5], while SYNC-PATH has the advantage of reducing the path length pertaining to synchronization by half, thus reducing the expected error as well. SYNC-PATH is also simple, general, and independent of the underlying routing protocol. Figure 7 depicts an example of path synchronization using the SYNC-PATH algorithm. In this example, we assume that a general routing protocol is used, and thus the

path P from src to dst is not known a priori. The synchronization trigger packet records nodes on the path as it travels across P.

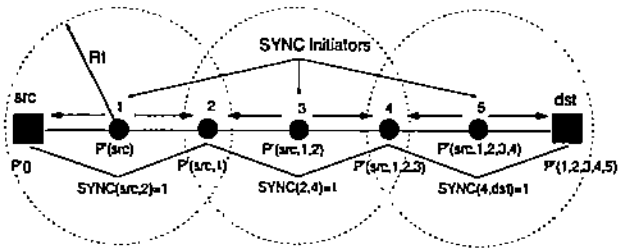


Fig. 7. Operation of the SYNC-PATH algorithm. All nodes except src and dst are cluster heads. All nodes on the path are assumed to be initially unsynchronized.

Observe that with loose synchronization, building routing tables for proactive routing approaches, such as DSDV, has to consider an extra parameter (synchronization) in addition to connectivity and cost. Similarly, reactive routing protocols, such as AODV or Directed Diffusion, have to compute paths only in synchronized directions. We assume, however, that SYNC-PATH will only be used in lightly-loaded networks, where queries occur rarely and in random parts of the network. In this case, SYNC-PATH will usually be invoked prior to data forwarding from the source, thus paving the way for data and relieving the routing protocol from the burden of searching for synchronized paths.

D. Inter-cluster Global Synchronization (SYNC-NET)

In this section, we develop a proactive time synchronization algorithm, SYNC-NET, for achieving relative synchronization in the entire network. Since global synchronization will usually be carried out in a heavily-loaded network, our goal is to construct strictly synchronized paths among every pair of nodes, and consequently between any source node and the observer[1] Applications that may use SYNC-NET are discussed in Section I. Node clustering is exploited as described later for scalability and fast convergence.

Our algorithm, SYNC-NET, strictly synchronizes the cluster head overlay and uses SYNC-IN to synchronize every cluster. The synchronization operation proceeds on a network clustered using any clustering scheme for ad-hoc networks, such as [21], [16], [22], [23], [14]. A scheme that results in well-distributed cluster heads in the network is ideal. This is because the resulting clusters can communicate with low power causing little interference, since cluster heads are non-neighbors (the LEACH protocol [17] may be ruled out because of its assumptions that may result in unexpected cluster head distribution). In addition, results from the analysis below can be applied in this case to prove that the network will asymptotically almost surely (a.a.s.) be synchronized using SYNC-NET. This will also be demonstrated via the experiments in Section IV.

Let C_{comm} be the set of nodes in the cluster head overlay. SYNC-NET will re-cluster the network using the set $V - C_{comm}$ only. This results in another cluster head overlay with a disjoint set of cluster heads C_{sync}, i.e., $C_{comm} \cap C_{sync} = \phi$. Since sensor networks are usually dense, we assume that obtaining two disjoint sets of cluster heads is possible (more details on the asymptotic conditions for achieving this are given in Section III-D.1). The two cluster head overlays have different roles. The first overlay, C_{comm}, is the overlay that will later be used for "time-aware" forwarding. Cluster heads in C_{comm} are also responsible for applying SYNC-IN for intra-cluster synchronization. Cluster heads in the overlay C_{sync} are used to synchronize the set C_{comm}, and therefore, other nodes in the network do not need to register themselves with cluster heads in C_{sync}.

The synchronization process proceeds as follows. Each cluster head $v \in C_{sync}$ discovers its neighbor cluster heads in C_{comm} using a transmission range R_t, where $R_c < R_t \leq R_m$. (R_t is also used for sending inter-cluster synchronization pulses.) A "neighbor" throughout this section refers to a node within a range R_t.[2] Knowledge of C_{comm} neighbors is used by each node in C_{sync} to determine when to terminate the execution of SYNC-NET, as described below. Each node v performs a few iterations during the execution of SYNC-NET. In the first iteration, v elects to become a synchronization initiator for its neighbors in C_{comm} with probability $0 < P_s \leq 1$ (say 5%). Thus, a synchronization initiator $v \in C_{sync}$ synchronizes a cluster head $u \in C_{comm}$ that covers an intersecting region with that of v, and all the cluster head neighbors of u in C_{comm}. The reason that we set nodes in C_{sync} to act as initiators probabilistically is to reduce the number of messages exchanged in the synchronization process. This is because a node in C_{sync} may redundantly act as an initiator, whereas if it waits for a few iterations, the nodes it is "responsible for" in C_{comm} may be synchronized with their neighbors as a result of other initiators in C_{sync}. In addition, starting with a small value of P_s allows gradual network synchronization and

[1]We assume the observer is included in the clustered network, but if this is not the case, the observer can be synchronized with the last node(s) on its routing path(s).

[2]Note that we assume that the multi-hop inter-cluster routing protocol will exploit a neighbor as the next hop in the inter-cluster routing path, which must be the case if R_t is the inter-cluster communication range.

thus lower interference. We will show in Section IV that this approach significantly reduces the number of synchronization initiators, and consequently the number of messages exchanged.

At the end of the first iteration, a cluster head that has elected to act as a synchronization initiator terminates its operation and exits SYNC-NET. A node $u \in C_{comm}$ that detects that it is currently synchronized with all its neighbors in C_{comm} broadcasts a "SYNC-DONE" message that it is done, and exits SYNC-NET. A cluster head $v \in C_{sync}$ that has not elected to act as an initiator in this iteration checks if all its neighbors in C_{comm} have sent "SYNC-DONE" messages. If so, v exits SYNC-NET. Otherwise, v doubles its P_s value, and proceeds to the next iteration. This process is repeated until P_s reaches 1. If the value of P_s of a node v reaches 1 before all the C_{comm} neighbors of v have sent "SYNC-DONE" messages to v, it will definitely act as an initiator. Note that when a node exits SYNC-NET, it ignores any newly received synchronization pulses. Figure 9 provides the pseudo-code for Algorithm SYNC-NET. It is clear that the algorithm is asynchronous, i.e., all nodes need not start executing it simultaneously for getting correct results. Figure 8 demonstrates the operation of SYNC-NET synchronization, where five cluster heads in C_{sync} are able to fully synchronize C_{comm} cluster heads with their neighbors.

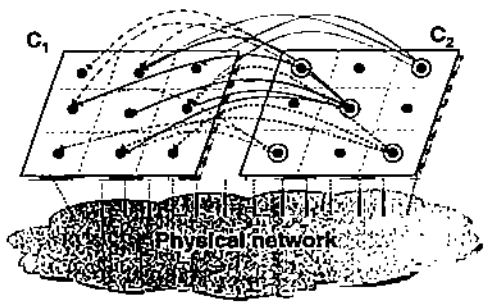


Fig. 8. The SYNC-NET protocol: Two cluster head overlays are constructed on top of the physical network. Five initiators in C_{sync} synchronize C_{comm}. A node in C_{comm} does not have any neighbors on the diagonal

1) Density Model: We now present the density model required to satisfy the conditions for forming two connected cluster head overlays. Assume that n nodes are uniformly and independently dispersed at random in an area $R = [0, L]^2$. Also assume that R is divided into N square cells of size $\frac{R_c}{\sqrt{2}} \times \frac{R_c}{\sqrt{2}}$ (thus $N = \frac{2L^2}{R_c^2}$), where a cell is an approximation of a cluster. This implies that every node in each cell can reach every other node residing in the same cell using a transmission range R_c. In [24], we presented a general density model for forming k connected cluster head overlays. This requires

Fig. 9. SYNC-NET: Inter-cluster Global Synchronization

// SYNC-NET uses two cluster head overlays
// C_{comm} and C_{sync}, $C_{comm} \cap C_{sync} = \phi$
// Execute the following at every node $v \in C_{sync}$
// P_s is the synchronization probability used by C_{sync} nodes

1. $S_{nbrs}[v] \leftarrow \{u : u \in C_{comm},\ distance(u, v) \leq R_t\}$
2. $Max\_iter = \lceil log_2 \frac{1}{P_s} \rceil + 1$
3. $iter = 0$
4. **REPEAT**
5. $iter \leftarrow iter + 1$
6. $r \leftarrow$ Uniform(0,1)
7. **IF** $r < P_s$
8. SYNC-2HOPS(R_t)
9. EXIT SYNC-NET
10. $S_{covered} = \{u : u \in C_{comm},$ u *has sent message* "$SYNC-DONE$"$\}$
11. **IF** $S_{covered} \neq S_{nbrs}$
12. $P_s \leftarrow min(P_s \times 2, 1)$
13. **UNTIL** ($iter = Max\_iter$ **OR** $S_{covered} = S_{nbrs}$)

a minimum cell occupancy of at least $k > 1$ nodes asymptotically almost surely (a.a.s.)[3] The following theorem provides the necessary conditions for minimum cell occupancy, and we use it in proving the correctness of our proposed algorithm (our algorithm requires that $k = 2$). Let $\eta(n, N)$ be a random variable that denotes the minimum number of nodes in a cell.

Theorem 1: For any fixed arbitrary $k > 0$, assume that n nodes are uniformly and independently distributed at random in an area $R = [0, L]^2$. Assume R is divided into N square cells, each of side $R_c/\sqrt{2}$. If $R_c^2 n \geq aL^2 ln\ N$ for some constant $a \geq 2$, $R_c \ll L$, and $n \gg 1$, then $lim_{n,N\to\infty} E[\eta(n, N)] = k$ iff $k \approx ln\ N$.

Proof. Refer to [24] for a complete proof.

2) Protocol Analysis:

Lemma 5: Each cell will a.a.s. have two distinct cluster heads, one in C_{comm} and the other in C_{sync}.

Proof. Assuming that Theorem 1 hold where $k = 2$, then every cell contains at least two nodes a.a.s., and consequently may easily contain two cluster heads, one for each cluster head overlay. The property holds a.a.s. because the nodes considered in constructing C_{sync} do not include the ones previously selected in C_{comm}. □

Lemma 6: When all nodes in C_{sync} terminate SYNC-NET, every node $u \in C_{comm}$ is synchronized with all its neighbors in C_{comm}.

[3]We regard a cell as an approximation of a cluster, and thus R_c is used to define the required density, and R_t is used to define connectivity.

Proof. Assume that R_t is selected such that it covers every cluster head in the complete neighborhood of cells around any cell A. The complete neighborhood around A is all the eight cells surrounding A (this can be ensured by enforcing a relation between R_t and R_c). Also assume that $\exists a_1 \in C_{comm}$, such that $S_{nbr}(a_1)$ is the set of neighbor cluster heads of a_1 in C_{comm}. We prove this lemma by contradiction. Assume that $\exists u \in S_{nbr}(a_1)$, such that $SYNC(a_1, u) = 0$. We assume that Theorem 1 holds (where $k = 2$), and therefore every cell contains two cluster heads (one in C_{comm} and the other in C_{sync}). There are two cases for u:

Case 1. The cell of node u is within the complete neighborhood of the cell of a_1. For example, as depicted in Figure 10, a_1 is in cell A, u can be one of $\{b_1, d_1, e_1, f_1\}$. In this case, the cluster head $a_2 \in C_{sync}$ can reach all of these nodes, and therefore can synchronize them with a_1, which is a contradiction.

Case 2. The cell of node u is not in the neighborhood of the cell of a_1 (cell A). For example, cell G in Figure 10 is one such case. Assume that a_1 and g_1 are neighbors, while a_2 and g_1 are not. However, there must exist another cluster head in a neighbor cell that belongs to C_{sync} (node d_2 in this example) which will not exit SYNC-NET until a_1 and g_1 are synchronized and send "SYNC-DONE" messages. This means that a_1 and g_1 will be synchronized, which is a contradiction. □

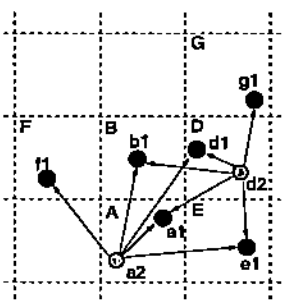


Fig. 10. Example of node synchronization using SYNC-NET. $\{a_1, b_1, c_1, d_1, e_1, f_1\} \subset C_{comm}$ and $\{a_2, d_2\} \subset C_{sync}$

Lemma 7: At every node $v \in C_{sync}$, SYNC-NET terminates in $O(1)$ iterations.

Proof. We assume that the clustering protocol is fast and $O(1)$ (e.g., [16], [22], [21]). The number of iterations taken until SYNC-NET terminates depends on P_s, since the operation continues until P_s reaches 1. Therefore, the number of iterations, N_{iter} can be computed as:

$$N_{iter} \leq \lceil log_2 \frac{1}{P_s} \rceil + 1 \qquad (1)$$

which is $O(1)$. For example, if P_s is 10%, then the number of iterations, N_{iter}, is 5. □

Lemma 8: SYNC-NET has an $O(1)$ message overhead per node in each cluster head overlay.

Proof. A node may elect to become a synchronization initiator in C_{sync} only once, and sends $O(1)$ synchronization pulses. A node to be synchronized in C_{comm} replies to synchronization pulses until all its neighbors are synchronized with it. The number of neighbors is $O(1)$ (which depends on the ratio R_t/R_c). Thus, every node in C_{comm} also sends at most $O(1)$ messages. □

Lemma 9: In the worst case, the synchronization accuracy of SYNC-NET is $O(\sqrt{N} \times q)$, where N is the number of cells in the network, and q is the accuracy obtained by the applied low-level synchronization mechanism.

Proof. Assume that the clustering mechanism used distributes cluster heads well across the network. We consider only the cluster head overlay, since communications proceed through it. The cluster head overlay can be approximated as a 2-D mesh network. Synchronization accuracy depends on the length of the path from the source to the destination, L_p, and the underlying low-level synchronization mechanism. In the worst case, L_p can be as long as the network diameter, which is $O(\sqrt{N})$. Therefore, the accuracy provided by SYNC-NET is $O(\sqrt{N} \times q)$. □

To quantitatively grasp the above lemma, consider a simple example of a sensor network with thousands of nodes. Assume $n = 10,000$ and $N = 100$ and RBS [5] is the underlying low-level synchronization scheme. RBS achieves an absolute accuracy per hop in the order of $q \approx 40\mu s$ on Berkeley sensor motes, as measured in [8]. Therefore, according to Lemma 9, SYNC-NET achieves an accuracy of $10 \times 40 \times 10^{-6} = 400\ \mu s$ on the longest expected path, in the worst case when errors add up.

IV. Performance Evaluation

In this section, we verify via simulations the properties of our proposed approaches for intra-cluster and inter-cluster synchronization.

A. Intra-cluster Synchronization

The primary advantage of the SYNC-IN algorithm is its fast convergence via exploiting knowledge of cluster members at each cluster head. We explore the two possibilities for selecting synchronization initiators that were discussed in Section III-B: (1) randomly, and (2) closest to the cluster head. In our experiment, we varied the number of nodes per cluster from 10 to 1000 to see how fast the algorithm terminates for different node densities. The transmission range (R_c) was fixed at 10 m. Thus, node density ranged from 0.1 nodes/m^2 to 10

nodes/m^2. Figure 11 illustrates that: (1) the number of iterations until SYNC-IN converges is less than 8 for different densities, which agrees with the result in Lemma 2, and (2) the number of iterations when the closest neighbors are selected as initiators is significantly lower than that when random initiators are selected. This is expected. However, selecting the closest neighbors as initiators adds an extra overhead on the cluster head for discovering neighbors at each power level (smaller than the cluster power level). This also requires additional message exchanges. Note that we have assumed in this experiment that there is an infinite number of transmission ranges below the cluster range. Practically, there will be only a few discrete usable power levels in any node. Therefore, the actual curve for using closest initiators will lie in the area between the two curves in Figure 11. The illustrated curve for using closest initiators simply gives a lower bound on the possible number of iterations.

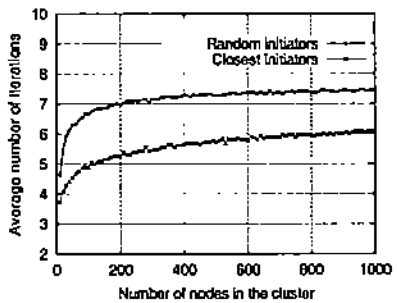


Fig. 11. Convergence of the SYNC-IN protocol

B. Inter-cluster Synchronization

Since SYNC-PATH is part of SYNC-NET, we evaluate the performance of our inter-cluster communication algorithms in the context of SYNC-NET. Throughout this section, we will use the term "neighbors" to refer to two cluster heads which can communicate using a transmission range $R_t > R_c$. We will only consider communications within the cluster head overlays, since intra-cluster communication was considered above. Two neighbor cluster heads can both belong to the same cluster head overlay, or belong to different overlays. As before, we use C_{comm} and C_{sync} to refer to the forwarding and synchronizing overlay cluster structures, respectively. We also use "node density" to refer to the number of nodes per cluster (cell), as defined in Section III-D.

In this section, we consider: (1) how the average number of neighbors varies as the transmission range grows, (2) how the average number of iterations until termination (convergence speed) varies as the node density increases, (3) how our approach of probabilistic synchronization initiation reduces the number of messages exchanged in the network, and finally, (4) what synchronization accuracy is achieved assuming uniform synchronization error distribution across the network and using RBS [5] as the underlying low-level synchronization protocol. We assume that n nodes are dispersed uniformly and independently at random within a 100 m$\times$100 m area. We fix R_c in most experiments. Changing R_c only results in changing the average number of cluster heads in C_{comm} and C_{sync}, and has no significant impact on the general performance of SYNC-NET.

To verify that the density model defined in Theorem 1 is sufficient for C_{sync} and C_{comm}, we carried out an experiment where the transmission range R_t is varied from double the cluster range R_c to four times the cluster range. R_c was selected to be 10 m. We performed this experiment for average node densities of 2.5 nodes/cell, 5 nodes/cell, and 10 nodes/cell, for 500, 1000, and 2000 nodes, respectively (we have also experimented with larger values). Figure 12 illustrates that the average number of neighbors in C_{comm} for each node in C_{sync} exceeds five, for all values of R_t. This number is important since it roughly indicates how many nodes will be synchronized if a node $v \in C_{sync}$ acts as a synchronization initiator. The figure also illustrates that the number of neighbors increases with the increase of R_t/R_c. Node density, as long as it satisfies Theorem 1 and is within certain bounds, does not appear to have as significant an impact on the results in this case, since the average number of neighbors is dominated by the ratio R_t/R_c.

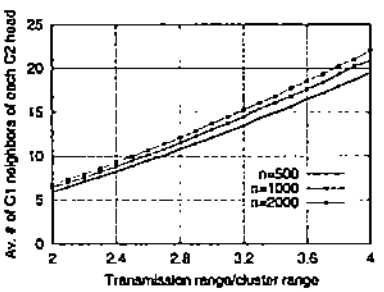


Fig. 12. Average number of neighbors from C_{comm} (labeled C_1) for each node in C_{sync} (labeled C_2)

Now we turn to the main focus of our proposed work: fast convergence. We compare the convergence speed of our SYNC-NET protocol (for different values of P_s) to that of a multi-hop TPSN [8]. We chose multi-hop TPSN as a representative of synchronization protocols whose

termination speed is dependent on the network diameter.[4] In TPSN, a reference node initiates synchronization by forming a hierarchy using message flooding. We use 4000 nodes in this experiment. The cluster range for SYNC-NET and TPSN neighbor discovery, R_c, varies from 3.5 m to 10 m. We plot the average number of iterations of TPSN for 100 random topologies. We also plot the maximum number of iterations of SYNC-NET for P_s = 0.01, 0.05, and 0.25 (which gives 8, 6, and 4 iterations respectively). We assume that a fast clustering protocol is used (i.e., an $O(1)$ protocol). Thus, we add 6 iterations to the SYNC-NET iterations (which is sufficient for protocols such as [16], [22], [21]), so that each SYNC-NET value represents the convergence speed of synchronization plus clustering. Figure 13 illustrates a significant difference in convergence speed between SYNC-NET and TPSN, especially for small R_c values. Note that TPSN was not slow in this experiment, due to the network organization in a 2-dimensional space and the uniform node distribution. In a 1-dimensional space, however, the number of iterations is expected to be $O(n)$ in the average case, which is much higher than the results specified in Figure 13. Observe, however, that this comparison is only for demonstration, since protocols like TPSN and Li-Rus [10] assume that the application needs to achieve a global time consensus in the network, which is not the goal of this work.

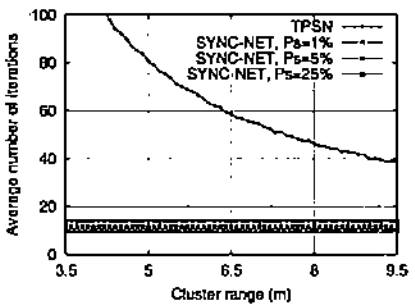
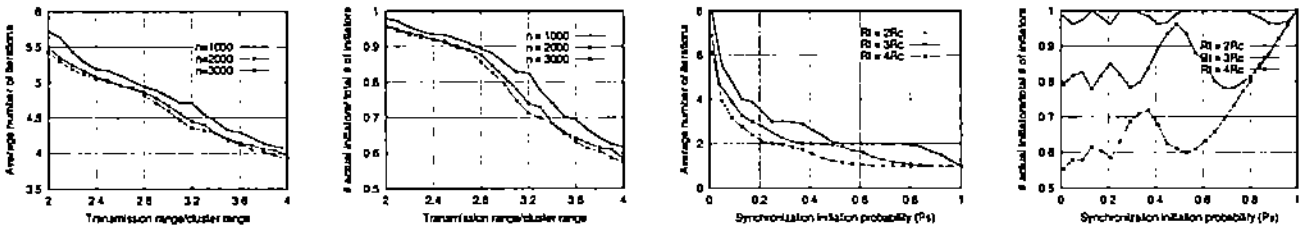


Fig. 13. Convergence speed of SYNC-NET and multi-hop TPSN

We now perform two experiments to verify Lemma 7. We compute the actual average number of iterations in these experiments to compare with the analytical upper bound and prove fast convergence. In both experiments, the transmission range R_t varies from $2R_c$ to $4R_c$, and R_c is fixed at 6 m. Experiments are performed for three values of n (the number of nodes): 1000, 2000, and 3000. This results in node densities that range from about 2 nodes/cell to 6 nodes/cell. Figure 14(a) shows that SYNC-NET terminates more rapidly as R_t grows relative to R_c. However, we also need to examine other aspects associated with longer transmission ranges, such as the number of exchanged messages. Figure 14(b) shows that the percentage of actual number of synchronization initiators out of the total number of viable initiators in C_{sync} is about 95% for $R_t = 2R_c$, and about 60% for $R_t = 3R_c$. This is a significant reduction in message exchange compared to the simple approach of blindly making every node in C_{sync} a synchronization initiator, since the percentage of non-participating nodes in C_{sync} reflects the percentage of reduced message overhead. As an example, consider the case where $n = 1000$, $R_t = 4R_C$, $R_c = 6m$, and $P_s = 5\%$. The number of cells $N = \frac{2L^2}{R_c^2}$, i.e., N=200 cells. Thus, the number of cluster heads in C_{sync} is approximately 200, assuming one cluster head exists per cell. Using the results in the model validation experiment, the average number of C_{comm} neighbors of each C_{sync} node is about 12. Also assume that the underlying low-level synchronization protocol is receiver-receiver (e.g., RBS [5]) and sends 10 synchronization pulses per initiator. If all the nodes in C_{sync} act as initiators, the expected number of message exchanges is: 200×10 (for initiation at C_{sync}) + 200×12 (for replies at C_{comm}) = 4400 messages. If only 60% act as initiators, this number is reduced to 2640 messages. Synchronization pulses sent by C_{sync} nodes can thus use the maximum available power level. This will not cause severe interference or energy consumption, since we synchronize the network "gradually" using a synchronization probability P_s.

In another experiment, we study the effect of the synchronization probability P_s on the convergence speed and message overhead of SYNC-NET. The probability P_s ranges from 0.01 to 1 in our experiments. The number of nodes n is set to 2000. The cluster range R_c is 6 m, while the transmission range R_t varies from $2R_c$ to $4R_c$. Figure 14(c) shows that (1) the average number of iterations until all the nodes in C_{comm} are synchronized with their neighbors is strictly less than the maximum specified by Lemma 7, and (2) as P_s increases, termination is faster as expected, since the synchronization probability goes to 1 quickly. This is not a desirable behavior, however, since more nodes in C_{sync} may redundantly send synchronization pulses. This is demonstrated in Figure 14(d), where smaller values of P_s generally result in a lower average number of initiators, and hence lower message overhead. The curves show more than one local minimum which means that each transmission range has a unique behavior with different synchronization probabilities. We have not considered

[4]We do not compare SYNC-NET with any approach in literature in other aspects, such as message complexity or perceived accuracy since our underlying assumptions, prospective applications, and objectives are completely different from all related work presented in Section VI



(a) Convergence speed for different Rt/Rc ratios

(b) % C_{sync} nodes that participated as sync initiators for different R_t/R_c ratios

(c) Convergence speed for different P_s values

(d) % C_{sync} nodes that participated as sync initiators for different P_s values

Fig. 14. Convergence speed and message overhead in SYNC-NET

the effect of interference in our simulations, which will indeed be magnified by sending simultaneous long-range synchronization pulses by neighboring nodes in C_{sync}. Therefore, we surmise that a small value of P_s (e.g., 5%) will help achieve both goals: fast termination and lower message exchange. This is because even for a small probability, the convergence speed is within practical bounds.

Finally, we consider the synchronization error propagated across the network as reports are transmitted from a source cluster head that is closest to the bottom left corner of the network area to an observer cluster head that is closest to the upper right corner. The number of nodes used is 1500 and 3000, while the transmission range R_t varies from $2R_c$ to $4R_c$. The cluster range R_c is fixed at 6 m. We use a simple error model: RBS low-level synchronization is employed for an absolute receiver-receiver synchronization error value of mean ± 40 μs introduced at every hop. This value was reported in [8] based on an implementation of RBS and experimental results on Berkeley sensor motes. Data is forwarded using greedy geographic routing for simplicity. Figure 15 illustrates the absolute error for different R_t/R_c ratios. Results show that the absolute error slightly increases using longer transmission ranges (i.e., fewer hops). This is not surprising in this scenario since the introduced error can be a positive or a negative value, and thus having a larger number of hops (i.e., smaller R_t/R_c) may increase the chance for error cancelation.

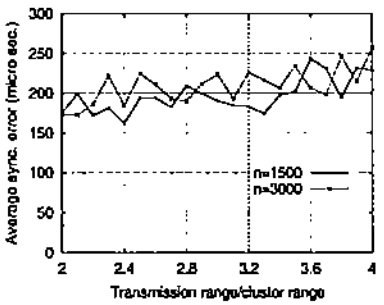


Fig. 15. Absolute average synchronization error for different transmission ranges

V. Discussion

In this section, we discuss several design and deployment issues pertaining to time synchronization in clustered ad-hoc sensor networks:

Sensitivity to the underlying clustering protocol: Our proposed algorithms are not dependent on the underlying clustering protocol. In fact, they can be applied to a non-clustered network, but will be less efficient in terms of message exchange and perceived accuracy. For example, in SYNC-NET, nodes can probabilistically elect themselves as initiators to synchronize the network. An initiator which was not synchronized with all its neighbors can perform a one-to-one synchronization with the remaining nodes. Node clustering provides a better organization for carrying out the synchronization process with more predictable asymptotic behavior. If the network is clustered only for synchronization, however, then the selected clustering mechanism should be fast in order not to dominate the overhead of the synchronization process. Examples of fast ($O(1)$) clustering protocols include [22], [21], [16], [17], [28].

Triggering synchronization: In many environments, clock skewness is continuously variable due to conditions such as temperature. In this case, SYNC-NET must be periodically triggered to re-compute relative synchronization in the network. SYNC-NET can be triggered at any node by timer expiration or by message exchange. For example, if the application can tolerate an error of up to 1 ms, and due to clock skewness, two clocks lose synchronization at a rate of 1 μsec per minute, then SYNC-NET should be triggered every 1000 minutes. It is practically difficult to compute the clock skewness for the entire network during the network operation.

However, each node can set a timer according to trigger synchronization taking into consideration the maximum "expected" skewness. When the timer expires, a cluster head in C_{sync} initiates a bounded-depth d-hop flooding to inform its d-hop neighbors in C_{sync} to start SYNC-NET. A node that hears this message starts the execution of SYNC-NET immediately, even if its timer has not yet expired. In clustered networks where clustering is re-triggered periodically to achieve certain goals, such as energy-efficiency or load-balancing, synchronization can be triggered separately from clustering, but should be triggered at least as frequently as clustering to maintain a synchronized clustered network.

Synchronization probability: We have shown how the synchronization probability, P_s, can limit the number of messages exchanged in the synchronization process. Throughout the paper, we have considered a constant P_s value for all nodes. A variable P_s can be used for attaining certain desirable properties. For example, P_s can be set to favor nodes with high degrees for faster convergence. Another option for P_s is to favor nodes with high remaining energy. This reduces the burden on nodes running out of battery, but requires that the clustering protocol provide "good" candidates in the cluster head overlay. We plan to explore these options in our future work.

Inter-cluster communication ranges: We have assumed in SYNC-NET that one inter-cluster transmission range will be used for communication within C_{comm} and C_{sync}. We have also recommended in Section IV that the synchronization pulses be sent by C_{sync} members using the maximum transmission range. This does not have to be the case for C_{comm} communications, since longer ranges will add more neighbors for each node, which increases interference and energy consumption, and requires more participation from nodes in the synchronization overlay to synchronize all of the nodes in C_{comm} with their neighbors. Thus, it is preferable that C_{comm} communications use the minimum transmission range that keeps the C_{comm} overlay connected, but is higher than R_c. This range can be computed using prior results such as [18], [19].

Sleeping nodes: Some applications require that nodes are not always awake to save energy. This may be problematic for cluster head overlay connectivity and synchronized path availability. This may also result in network partitioning. To avoid these problems, a node that belongs to C_{comm} should never go to the sleeping mode until it is no longer a cluster head. In addition, the application should control the sleep/wake up process to ensure that the set of active nodes at any time satisfy the density model in Theorem 1.

Fault-tolerance: Networks deployed in hostile environments, such as volcanic areas or military fields, may experience unexpected node failures. This might hinder communication and synchronization if the failed nodes are in the cluster head overlay. This is somewhat mitigated by the fact that applications that use clustered networks usually re-cluster the network periodically to maintain connectivity and adjust to changing network conditions due to dispersion of new nodes or failure and energy depletion of others. Fault tolerance can be achieved by maintaining backup independent cluster head overlays, and not just one as we use in SYNC-NET. These backup overlays can also be synchronized using the synchronization pulses generated by the synchronization overlay initiators. The number of independent overlays that can be constructed, however, is limited by the node density and distribution in the network [24].

Applicability: Our proposed framework is applicable to any wireless network setting that requires scalable and fast convergence. We have explained our framework within the context of sensor networks to show one viable application. Several other types of distributed systems running on wireless ad-hoc nodes can utilize our framework, though. These include several data dissemination peer-to-peer, and network monitoring applications. Examples are BitTorrent or KaZaA or network monitoring tools, running on handheld or laptop devices in a wireless network in ad-hoc mode.

VI. Related Work

The Reference Broadcast Synchronization (RBS) [5] is a low-level receiver-receiver protocol that aims at high accuracy and low cost. RBS does not need any infrastructure support and can achieve an accuracy of 6 μs on sensors with a 4 MHz clock. RBS, however, suffers from a number of limitations when applied in sensor networks. First, for single-hop synchronization, RBS assumes that all the receivers are within the same broadcast range. Second, the receivers (of the generated sync pulses) are synchronized among themselves, while the initiator of these pulses is not synchronized. Third, for multi-hop synchronization, RBS assumes that certain receivers will belong to multiple intersecting synchronization regions, and thus synchronization information can be propagated as packets are routed through nodes in the intersection areas, which may not always be feasible. Romer [6] provided a synchronization mechanism for ad-hoc networks that assumes uni-directional links and achieves 1 ms accuracy. Cristian [12] proposed a probabilistic synchronization approach where synchronization is achieved by sending multiple packets until the error is bound by a pre-defined constant. The basic

TPSN [8] and Ping's technique [11] use sender-receiver synchronization for a higher accuracy than RBS, assuming that timestamping can be done at the MAC layer. CesiumSpray [9] uses receiver-receiver synchronization and applies a hierarchical structure to achieve scalable synchronization. It assumes that the network contains a number of distributed GPS-enabled sensors which can contact the GPS satellite (the top of the hierarchy). The Network Time Protocol (NTP) [3] is a sender-receiver synchronization approach widely deployed in the Internet and has proved to be scalable and robust. In NTP, a hierarchy of time servers is deployed and receivers consult with their parent servers to adjust their clocks. These protocols provide both low-level and high-level synchronization.

In addition to these, several protocols were proposed for high-level synchronization. The idea of *virtual* clocks was proposed in [2]. Virtual clocks are used for synchronization if the absolute (physical) time is not necessary, such as in the ordering of events. This work, however, assumes that message reception preserves the event ordering. The post-facto synchronization mechanism [7] is proposed for systems where events do not occur too often, and thus synchronization is performed only when necessary. The Classless Time Protocol (CTP) [25] formulates the clock offset problem as an optimization problem and gives a distributed algorithm to reach the optimal solution. High-level synchronization was also proposed in [8] (which we refer to as multi-hop TPSN). This work proposes a hierarchical approach for high-level synchronization using message flooding. Building the hierarchical structure can be $O(n)$ depending on the topology. Multi-level RBS [5] also relies on RBS as the underlying low-level synchronization mechanism and assumes that intersecting regions have nodes that may carry out inter-regional synchronization. This also requires time-aware routing protocols. Li and Rus [10] assume that all network nodes need to agree on a global clock value (which is different from our goal) and propose centralized and distributed approaches to reach this goal. The centralized approach does not scale, while the distributed (diffusion-based) approach is $O(n)$, which does not satisfy our goal of fast convergence.

Table II classifies time synchronization research relevant to ad-hoc and sensor networks according to goals and approach. The multi-hop RBS protocol [5] relies on the presence of nodes in the intersection areas of regions, and thus does not incur any extra message or processing overhead for the synchronization process. The complexity, however, is pushed to the routing protocol. Global synchronization is not necessarily achieved in this case as was described in Figure 3. According to Gupta [19], connectivity in an ad-hoc network is achieved if $r^2 = \frac{c log\ n}{n}$, where n is the number of nodes, r is the transmission range, and c is a constant. Using this formula, the number of neighbors of any node is $O(log\ n)$ if the network is connected. This is the message complexity of certain protocols, such as the Diffusion-based approach [10] and multi-hop TPSN [8], in which a node receives at least one message from each of its neighbors, and forwards it.

Several distributed clustering approaches have been proposed for mobile ad-hoc networks and sensor networks. In one approach, protocols are weight-based, i.e., clustering is according a certain parameter (weight) or a number of parameters, such as node degree or residual energy (e.g., [26], [16], [22], [17]). Examples of weight-based approaches include the Distributed Clustering Algorithm (DCA) [27], the Weighted Clustering Algorithm (WCA) [26], Estrin et al. [28], ACE [22], HEED [16], and LEACH [17]. Another approach is to cluster the network by selecting a dominating set, such as [29], [21], [23]. In a third approach, protocols are heuristic-based, e.g., cluster the network using node identifiers, e.g., [13]. Finally, a number of approaches construct a clustered network in order to optimize routing while supporting mobility, e.g., [14].

VII. Conclusion

In this work, we proposed a distributed, high-level time synchronization framework for multi-hop sensor networks that integrates node synchronization with node clustering for scalability and fast convergence. Our framework serves the two major classes of network applications, namely, source-driven and data-driven network applications. We defined synchronization regions as clusters where two-hop communication can take place through a cluster head. We designed fully distributed protocols for intra-cluster synchronization (SYNC-IN), and inter-cluster synchronization, including global path synchronization (SYNC-PATH) and global network synchronization (SYNC-NET). Our protocols focus on the fact that for most applications, fast convergence and scalability are the main objectives, and coarse-grained granularity is sufficient at the global scale. We analyzed our proposed approaches, and evaluated them via simulations.

For intra-cluster synchronization, results show that a 2-hop region can be synchronized in less than 8 iterations using a receiver-receiver low-level synchronization protocol. For inter-cluster synchronization, results show that the proposed analytical density model is easily achieved in moderately dense networks, where the expected number of nodes per cell exceeds two. Results

TABLE II

CLASSIFICATION OF SYNCHRONIZATION PROTOCOLS FOR AD-HOC AND SENSOR NETWORKS. LL, AND HL DENOTE LOW-LEVEL AND HIGH-LEVEL SYNCHRONIZATION, RESPECTIVELY. RR AND SR DENOTE RECEIVER-RECEIVER AND SENDER-RECEIVER APPROACHES, RESPECTIVELY.

Protocol	Classification	Primary Goal	Major assumptions	Convergence speed	Message overhead
RBS [5]	LL, RR	High accuracy	All nodes can reach each other in one hop	$O(1)$	Sender: m, receiver: 1
TPSN [8], Ping [11]	LL, SR	High accuracy and minimum overhead	Timestamping can be done at the MAC layer	$O(1)$	Sender: 1
Romer [6]	LL, SR	Temporal ordering	Nodes may be mobile	$O(1)$	$O(1)$ per node
Cesium Spray [9]	LL/HL, RR	Accuracy and scalability	A few GPS-enabled nodes are randomly scattered	LL: $O(1)$, HL: $O(log\ n)$	Sender: 1
Diffusion-based [10]	HL	All nodes have to agree on one time value	Only coarse-granularity is required	$O(n)$	$O(log\ n)$ per node
Multi-hop RBS [5]	HL, RR	Time-aware multi-hop communications	Nodes exist on the intersection areas of regions, and routing is time-aware	N/A	N/A
Multi-hop TPSN [8]	HL, SR	All nodes have to agree on one reference time value	$\exists$ at least one capable reference node, i.e., GPE-enabled	$O(n)$ to build the hierarchy	$O(log\ n)$ per node for flooding
Post-facto [7]	HL	Synchronizing 1-hop regions	Rare occurrence of events	Depends on the underlying LL	Depends on the underlying LL
Lamport [2]	HL	Event ordering	Messages reflect the ordering of events	$O(n)$	$O(1)$
SYNC-NET (this work)	HL	Fast global synchronization, and minimizing message exchange	The node density allows 2 independent (node-disjoint) network clusterings	$O(1)$	$O(1)$ per node

also indicate that by using high transmission power for sending synchronization pulses, and by gradual network synchronization (through a probability P_s), message overhead can be significantly reduced.

As a result of using node clustering, energy-efficiency can be achieved since periodic re-clustering distributes energy consumption in the network, and thus prolongs the network lifetime. Our framework is useful for a number of ad-hoc wireless network settings. We have presented it in the context of sensor networks to provide a viable application in which this framework is important. We plan to implement our proposed protocols on a sensor testbed and carry out small scale experiments as a proof-of-concept. We also plan to study the effect of of node distribution in the network, and the impact of variable probability P_s values.

REFERENCES

[1] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, 2001.

[2] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[3] D. L. Mills, "Network Time Protocol (Version 3): Specification, Implementation and Analysis," *RFC 1305*, March 1992.

[4] J. Elson and K. Romer, "Wireless Sensor Networks: A new Regime for Time Synchronization," in *Proceedings of HotNets-I*, October 2002.

[5] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time synchronization Using Reference Broadcasts," in *Proceddings of OSDI*, 2002.

[6] K. Romer, "Time Synchronization in Ad-hoc Networks," in *Proceedings of MobiHoc*, October 2001.

[7] J. Elson and D. Estrin, "Time Synchronization for Wireless Sensor Networks," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, April 2001.

[8] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync Protocol for Sensor Networks," in *Proceedings of ACM SenSys*, November 2003.

[9] P. Verissimo, L. Rodrigues, and A. Casimiro, "CesiumSpray: A Precise and Accurate Global Time Service for Large-scale Systems," *Special Issue on the Global Time in Large-scale Distributed Real-time Systems, Journal of Real-time Systems*, vol. 12, no. 3, November 1997.

[10] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," in *Proceedings of IEEE INFOCOM*, March 2004.

[11] S. Ping, "Delay Measurement Time Synchronization for Wireless Sensor Networks," *Intel Research, IBR-TR-03-013*, June 2003.

[12] F. Cristian, "Probabilistic Clock Synchronization," *Distributed Computing*, pp. 146–158, 1989.

[13] C. R. Lin and M. Gerla, "Adaptive Clustering for Mobile Wireless Networks," in *IEEE J. Select. Areas Commun.*, September 1997.

[14] B. McDonald and T. Znati, "Design and Performance of a Distributed Dynamic Clustering Algorithm for Ad-Hoc Networks," in *Annual Simulation Symposium*, 2001.

[15] V. Kawadia and P. R. Kumar, "Power Control and Clustering in Ad Hoc Networks," in *Proceedings of IEEE INFOCOM*, April 2003.

[16] O. Younis and S. Fahmy, "Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach," in *Proceedings of IEEE INFOCOM*, March 2004.

[17] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, October 2002.

[18] S. Shakkottai, R. Srikant, and N. Shroff, "Unreliable sensor grids: Coverage, connectivity and diameter," in *Proceedings of IEEE INFOCOM*, March 2003. [Online]. Available: citeseer.ist.psu.edu/shakkottai03unreliable.html

[19] P. Gupta and P. R. Kumar, "Critical Power for Asymptotic Connectivity in wireless Networks," *Stochastic Analysis, Control, Optimizations, and Applications: A Volume in Honor of W.H. Fleming, W.M. McEneaney, G. Yin, and Q. Zhang (Eds.), Birkhauser, Boston, 1998*, 1998.

[20] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, 2000.

[21] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, "Max-Min D-Cluster Formation in Wireless Ad Hoc Networks," in *Proceedings of IEEE INFOCOM*, March 2000.

[22] H. Chan and A. Perrig, "ACE: An Emergent Algorithm for Highly Uniform Cluster Formation," in *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, January 2004.

[23] S. Banerjee and S. Khuller, "A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks," in *Proceedings of IEEE INFOCOM*, April 2001.

[24] O. Younis and S. Fahmy, "Robust Communications for Sensor Networks in Hostile Environments," in *the Twelfth International Workshop on Quality of Service (IWQoS'04)*, June 2004.

[25] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, August 1999.

[26] O. Gorewitz, I. Cidon, and M. Sidi, "Network Time Synchronization Using Clock Offset Optimization," in *Proceedings of ICNP*, November 2003.

[27] M. Chatterjee, S. K. Das, and D. Turgut, "WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks," *Cluster Computing*, pp. 193–204, 2002.

[28] S. Basagni, "Distributed Clustering Algorithm for Ad-hoc Networks," in *International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, 1999.

[29] F. Kuhn and R. Wattenhofer, "Constant-Time Distributed Dominating Set Approximation," in *ACM Symposium on Principles of Distributed Computing (PODC)*, July 2003.